7 years ago I started comma.ai with a simple idea.

1. Gather tons of human driving data, state action pairs: `(S_t, A_t)`
2. Train a supervised model `f(S_t) -> A_t`
3. Drive cars with that model.

The exact original formulation was a model that predicts steering angle from image, then used a
PID loop to bring the wheel to that desired angle.

`f_steerangle(img_t) -> steerangle_t`

This turns out not to work, it couldn't even drive straight on highways. It would drive for maybe 1
seconds, but then error would accumulate and it would drift to one side of the lane or the other
(funny enough, it did show reluctance to cross the lane line, but it was unusable as an ADAS
system)

---

comma's first solution was a model that predicted lane position.

`f_lane(img_t) -> (left_lane_pos_t, right_lane_pos_t)`

While that alone couldn't drive a car (especially not around turns), it functioned as a unbiased
correction for the steering angle model, where `α` is the correction factor.

`(f_steerangle(img_t) - α*f_lane(img_t).mean()) -> steerangle_t`

This was basically shipped in the first version of openpilot.

One major issue this struggled with was ground truthing the lane line model. Unlike steering an
which has a simple sensor to measure it, "lane lines" don't have a clear definition. They broke th
end-to-endness of the system.

We referred to lanes as the "original sin" of comma, and tried really hard to remove them. I'm sa
say that there's still lanes in our ground truthing stack today, but we have made amazing strides
removing them, to the point that openpilot in 2020 could drive on a dirt road without any lane lin

However, the removal of lanes was done with a whole bunch of other hand coding. We have
extended this to removing explicit use of cars with experimental mode, but some of our hand co
assumptions break down a bit more in the longitudinal case vs the lateral case.

Funny enough, things have come full circle, and we think we have a solution to behavioral cloni
will explain the problem as I best understand it, and leave the solution as an exercise to the rea

Imagine running the steering angle model over time. At each time step, any model makes `ε` er

```
f_steerangle(img_t0) + ε_t0 -> steerangle_t0
f_steerangle(img_t1) + ε_t1 -> steerangle_t1
f_steerangle(img_t2) + ε_t2 -> steerangle_t2
f_steerangle(img_t3) + ε_t3 -> steerangle_t3
...
```

This model is easy to train, and can achieve very low losses on a holdout set. However, it won't drive a car, and that's due to the `ε` errors altering the next image. Note that the errors don't alt the next image in either train or test, but on the road it looks like:

```
f_steerangle(img_t0) + ε_t0 -> steerangle_t0
f_steerangle(img_t1') + ε_t1 -> steerangle_t1
f_steerangle(img_t2'') + ε_t2 -> steerangle_t2
f_steerangle(img_t3''') + ε_t3 -> steerangle_t3
...
```

If it is driving well depends on how far `img_t3'''` is from `img_t3`, which depends on what `ε_t0 + ε_t1 + ε_t2 + ε_t3 + ...` looks like in the limit.

Are the `ε` correlated? In the best case they aren't, but in practice they almost always are. And even if they aren't correlated, that error *still* grows unbounded. You need them to be **anti-correlated**. You need the limit of that sum to be 0.

You need an estimator of accumulated episilon. Above, we use `f_lane(img_t).mean()`, but imagine the generic form.

```
f_steerangle(img_t0) + ε_t0 -> steerangle_t0
f_steerangle(img_t1') + ε_t1 - α*ε_t0  -> steerangle_t1
f_steerangle(img_t2') + ε_t2 - α*(ε_t1 - α*ε_t0) -> steerangle_t2
f_steerangle(img_t3') + ε_t3 - α*(ε_t2 - α*(ε_t1 - α*ε_t0))-> steerangle_
...
```